```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TOL 0.0000001

/* Problem # 2 Using Successive OverRelaxation Method

*/

double dblAbs(double num){
    num = sqrt(num*num);
    return num;
}

int main()
{

    double a[13][13] = {
    { 1.0250, -0.4000,  0.0000,    0.0000, -0.6250,  0.0000,    0.0000,   0.0000,  0.0000,  0.0000,    0.0000,   0.0000,  0.0000},
    {-0.4000,  1.9135, -0.2998,  0.0000,  0.0000, -1.0338, -0.1799,  0.0000,  0.0000,  0.0000,    0.0000,   0.0000,  0.0000},
    { 0.0000, -0.2998,  2.1415, -0.4417,  0.0000,  0.0000, -1.4000,  0.0000,  0.0000,  0.0000,    0.0000,   0.0000,  0.0000},
    { 0.0000,  0.0000, -0.4417,    1.0077,  0.0000,  0.0000,    0.0000, -0.5660,  0.0000,  0.0000,    0.0000,   0.0000,  0.0000},
    {-0.6250,  0.0000,  0.0000,    0.0000,  2.0500, -0.8000,    0.0000,   0.0000, -0.6250,  0.0000,    0.0000,   0.0000,  0.0000},
    { 0.0000, -1.0338,    0.0000,   0.0000, -0.8000,  3.8097, -0.4401,    0.0000,  0.0000, -1.0000, -0.0595,    -0.4764,  0.0000},
    { 0.0000, -0.1799, -1.4000,    0.0000,  0.0000, -0.4401,    4.0492, -0.7067,  0.0000,  0.0000,    0.0000,  -1.1159, -0.2067},
    { 0.0000,  0.0000,    0.0000, -0.5660,  0.0000,  0.0000, -0.7067,    2.2161,  0.0000,  0.0000,    0.0000,   0.0000, -0.9433},
    { 0.0000,  0.0000,    0.0000,   0.0000, -0.6250,  0.0000,    0.0000,   0.0000,  1.0250, -0.4000,    0.0000,   0.0000,  0.0000},
    { 0.0000,  0.0000,    0.0000,   0.0000,  0.0000, -1.0000,    0.0000,   0.0000, -0.4000,  2.0667, -0.6667,    0.0000,  0.0000},
    { 0.0000,  0.0000,    0.0000,   0.0000,  0.0000, -0.0595,    0.0000,   0.0000,  0.0000, -0.6667,    1.3788,  -0.6526,  0.0000},
    { 0.0000,  0.0000,    0.0000,   0.0000,  0.0000, -0.4764, -1.1159,    0.0000,  0.0000,  0.0000, -0.6526,    2.5098, -0.2650},
    { 0.0000,  0.0000,    0.0000,   0.0000,  0.0000,  0.0000, -0.2067,   -0.9433,  0.0000,  0.0000,    0.0000,  -0.2650,  1.4150},
    };


    double b[13] = {2, 0, 0, 0, 4, 0, 0, 0, 2, 0, 0, 0, 0};

    double x_old[13] = {0, 0, 0, 0, 0, 0, 0,0, 0,0,0,0,0};
    double x_new[13] = {0, 0, 0, 0, 0, 0, 0,0, 0,0,0,0,0};
    double err[13];
    double u0 = 100;
    double w = 0.01;
    double w0 = 1;
    double dw = 0.05;
    double wn = 1.5;
    double dummy, m;
    int method = 2;                         // Methods: 0: Jacobi Method, 1: Gauss-Seidel, 2: SOR

    int total, col, row, k, i, j;
    double maxerr, diff, L2;

    total = sizeof(a)/sizeof(a[0][0]);      //
    col = sizeof(a[0])/sizeof(a[0][0]);     // Determines the number of rows and columns
    row = total/col;

    for(i=0; i<row; i++){
        b[i] = b[i]*u0;
    }

    //Perform Partial Pivoting;
    for(i=0; i<row-1; i++){
        //comparison to select the pivot
        for(j = i+1; j<row; j++) {
            if (dblAbs(a[j][i]) > dblAbs(a[i][i])) {
                for(k=0; k<col; k++){
                    dummy = a[i][k];
                    a[i][k] = a[j][k];
                    a[j][k] = dummy;
                }
                dummy = b[i];
                b[i] = b[j];
```

```
                b[j] = dummy;
            }
        }
        //Make an Upper Triagular Matrix
        for(j = i+1; j<row; j++) {
            m = a[j][i]/a[i][i];
            for(k = 0; k<col; k++){
                a[j][k] = a[j][k] - m*a[i][k];
            }
            b[j] = b[j] - m*b[i];
        }
    }

    if (method < 2) {
        wn = w0;
    }else {
        printf("  w  Total Iterations            \n");
        printf("\n");
    }

    for (w = w0; w <=wn+dw; w = w + dw) {
        maxerr = 100;           // So that it will go to while loop
        k = 0;

        for (i = 0; i < row; i++){
            x_old[i] = 0;
            x_new[i] = 0;
        }

        while (maxerr > TOL) {
            k++;                //Iteration number

            for (i = 0; i < row; i++) {
                /** Using Jacobi method only **/

                // Update values of x_new
                x_new[i] = b[i];
                for (j = 0; j < col; j++) {
                    if( i != j) {
                        x_new[i] = x_new[i] - a[i][j] * x_old[j];
                    }
                }
                x_new[i] = (x_new[i])/a[i][i];


                /** with SOR Method **/
                if(method == 2) {
                    x_new[i] = w*x_new[i] + (1-w)*x_old[i];
                }
                diff  = x_new[i] - x_old[i];
                err[i] = sqrt(diff*diff);       // Get absolute value of convergence
criterion vectors

                /*** with Gauss-Seidel Method ***/
                if (method > 0) {
                     x_old[i] = x_new[i];            // Comment out to use Jacobi method
                }
            }

            L2 = 0;                             // Linfinity - norm
            for (i = 0; i < row; i++) {
                x_old[i] = x_new[i];            // Update values of x_old
                if (err[i] > L2) {
                    L2 = err[i];
                }
            }
            maxerr = L2;

            if (method <2) {
                printf("Iteration number = %d  with tolerance = %0.9lf \n",k,maxerr);
            }

        }

        if (method < 2) {
            printf("\nThe answers are: \n");
            for (i  = 0; i < row; i++){
                printf("x_new[%d] = %0.9lf\n", i+1,x_new[i]);
            }
            printf("\nTotal Number of Iterations = %d\n", k);
        }else {
            printf("% 0.2lf       %d       \n",w,k);
        }
    }
```

```
154
155         return 0;
156     }
157
158
```